Solution-Schicht — Product Summary

> **In einem Satz:** Wir verpacken den vorhandenen Workflow-Stack in eine **Solution-Schicht**, mit der ein Business-Kunde seine wiederkehrenden Aufgaben selbst **anlegt, einstellt, testet, startet und ansieht** — ohne Code pro Kunde.
>
> Volldokument: `2026-06-STEP1-customer-workflows-solutions-architektur.md` · Mockup: `2026-06-STEP1-solutions-mockup.html`

Die Idee dahinter

Klassische Software ist von gestern. Statt pro Problem eine Applikation zu *schreiben*, **konfigurieren** wir komplexe Use Cases aus vorhandenen Bausteinen — genau das, was eine AI heute schon tut, wenn sie assistiert: Absicht beschreiben → System komponiert das Ergebnis.

Die Solution-Schicht **produktisiert** das: Der Kunde beschreibt seinen Use Case, der Workflow-Agent komponiert den Graph, das Ergebnis läuft **dauerhaft, geplant und governt**. Massgeschneidert ohne Massanzug-Kosten, Tage statt Monate, Self-Service statt Vendor-Backlog. **Grenzkosten eines neuen Use Cases → fast null.**

Ehrlich zu Ende gedacht: Es verschwindet nicht *alle* Software — die **Bausteine** und einige **tiefe Features** bleiben Code. Aber die vielen massgeschneiderten Einzellösungen, die man bisher pro Kunde gebaut hat, werden zu **Konfiguration**. Genau das ist «konfigurieren statt coden» — der Hebel, der Kunden aufs nächste Level bringt.

Das SaaS-Asset: Solution vs. Feature

PORTA hat heute zwei Enden: den technischen **Workflow-Editor** und das **Ops-Dashboard**. Es fehlt die **Mitte** — die kundentaugliche Schicht. Genau das ist das Produkt-Asset:

- **Solution = konfigurierter Workflow (Daten).** Aus Bausteinen komponiert, vom Kunden per Settings eingestellt. Time-to-value: **Tage**. Skaliert **ohne** Engineering pro Kunde.
- **Feature = Code.** Eigene Datenmodelle + tiefe, eigene UI (z. B. `lawyer`). Time-to-value: **Wochen**. Nur wo nötig.

Leitsatz: Solution-first. Ein Feature bauen wir nur, wenn die Solution-Schicht es nachweislich nicht trägt. So wird «konfigurieren statt coden» der Default — und das macht aus «wir bauen pro Kunde» ein skalierendes SaaS.

	Solution (Daten)	Feature (Code)
Datenmodell	nutzt bestehende	eigenes
UI	generische «Lösungen»-Seite	eigene, opinionated
Erstellung	Settings + Graph	Engineering
Beispiele	Pling-Reporting, SelectLine→RMA, PWG	lawyer, commcoach, trustee

Die vier Schichten

	Schicht	Inhalt	Status
L4	Solution Surface	Kundentaugliche UX: «Lösungen»-Seite (Liste · Katalog/AI · Detail mit Tabs)	**NEU**
L3	Solution Definition	Use Case → konfigurierter Workflow: Settings-Schema + Werte + Trigger + Output	**NEU**
L2	Composition	Graph: AutoWorkflow/Version/Run, Editor, Templates	vorhanden
L1	Plattform-Toolbox	Connectors, Actions/Nodes, AI, Neutralisierung, Scheduler, RBAC	vorhanden

→ **Unsere Arbeit liegt in L3 + L4.** L1/L2 nutzen wir unverändert.

Grundlage: die Kunden-Cases

Wir bauen aus realen Cases, nicht aus einer Idee. Sie sind die **Akzeptanzkriterien**: trägt der Entwurf den Fall, ist er richtig.

Case	Bedürfnis	Settings (= Kundenspezifika, Daten)	Output
Pling (Leit-Case)	5 Buchhaltungen → 6 PDFs, rollenbasiert mailen, monatlich	5 Instanzen, Konten-Ranges, Cron, Empfänger-Rollen	6 PDFs + Mails
SelectLine→RMA	Rechnungen täglich ins RMA buchen	Quelle/Ziel, Mapping	Sync-Log
PWG	Mietzins-Belege verarbeiten + Antwort	Ordner, Mandant, Empfänger	Tabelle + Mail
Lawyer	Mandatsvorbereitung + Dashboards	—	= **Feature**, nicht Solution

Was die Cases erzwingen: **Multi-Instanz-Fan-out** (Pling: 1 Solution über 5 Instanzen) und zwei **generische** Bausteine (`rbac.queryUsersByRole`, Trigger-`accessControl`) — wiederverwendbar, kein Kunden-Code.

L3 — Datenmodell (kurz)

Solution = **dünne Erweiterung von** `AutoWorkflow` + Settings-Record (keine Parallel-Welt → Scheduler/Runs/Versioning greifen unverändert). Felder:

`workflowRef` · `settingsSchema` (auto aus `trigger.form`/exposed Params) · `settingsValues` (inkl. **Instanz-Set**) · `triggerPolicy` (+ `accessControl.requiredRoles`) · `outputBinding` (`file | table | summary`) · `customerFacing`.

L4 — UI (kurz)

Eine «Lösungen»-Seite pro Feature (Start: Trustee) — kein Seiten-Wildwuchs:

- **Liste** (aktiv/inaktiv, letzter/nächster Lauf)
- **Katalog/Neu** (Vorlage wählen oder per AI beschreiben)
- **Detail** mit Tabs: **Einstellungen · Testlauf · Läufe · Ausgabe**

Settings-Formular wird aus dem `settingsSchema` gerendert (FrontendType). «Im Editor öffnen» bleibt die Escape-Hatch nach L2.

Bauen — erster Schnitt

1. `AutoWorkflow` um `settingsSchema` + `customerFacing` + Instanz-Set erweitern (+ Settings-Record, Migration).
2. 2 System-Templates: «Systeme synchronisieren» (SelectLine→RMA) + «Periodisches Reporting» (Pling).
3. Generische `SolutionsView` im Trustee: Liste + Settings + Testlauf + Aktivieren + Läufe.
4. Generische Bausteine: `rbac.queryUsersByRole`, Trigger-`accessControl`, run-level `testMode` (Kommunikation im Testlauf unterdrücken).

→ End-to-end belegt an **SelectLine→RMA** und **Pling**.